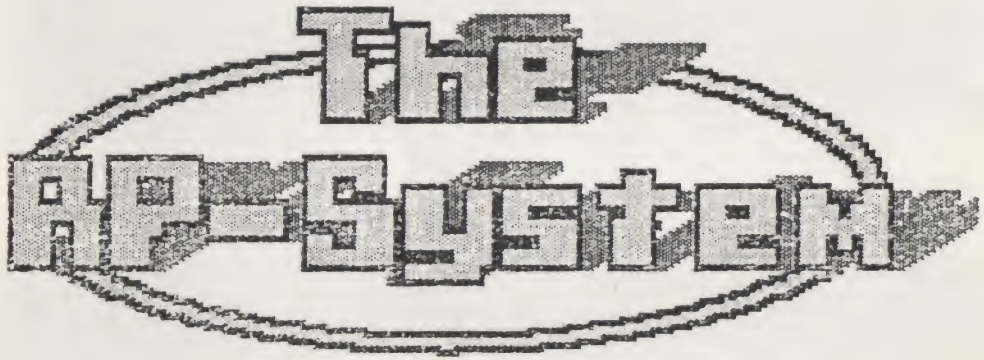

Modedeal Co. Ltd.

Wichtiger Hinweis:
Diese Diskette ist nicht kopiergeschützt. Der rechtmäßige Erwerber ist berechtigt, sich eine Sicherheitskopie anzufertigen.

Jede Weitergabe, unentgeltlich, als Tausch- oder Handelsware ist ohne die Einwilligung von Modedeal Co. Ltd. untersagt und wird unnachsichtlich gerichtlich verfolgt.

Turbo Speed Basic und RP-System dürfen als Teil eines vom Erwerber erstellten Programmes in jeder Form vertrieben werden, sofern sie Teil des Programmes sind und nicht separat auf dem Datenträger enthalten sind. Das Programm, das Turbo Speed Basic oder RP-System enthält, muß mit dem PGMSAV-Befehl und ohne Änderung abgespeichert worden sein.

Alle Programme wurden sorgfältig getestet. Trotzdem kann es bei Programmen dieser Größenordnung passieren, daß Fehler übersehen wurden. Modedeal Co. Ltd. macht darauf aufmerksam, daß bei Fehlern im Programm keinerlei Schadenersatz gewährt werden kann.

Bücher und Software sind vom Umtausch oder Rückgaberecht ausgeschlossen.

Congratulations; you have bought a program that will soon be able to help you produce professionally looking games in a much shorter time than before.

But at first, I'll have to tell you, that it is much harder for me to write the english instructions, than to program the entire system. The RP-System is made in Germany. So I hope you always can understand, even if there are some words or sentenses that do not sound general for you.

The RP-System for the Commodore 64 is a memorybased extension. When you have programmed a game, you can save it together with the RP-System to disk or cassette and may even sell it, as far as you saved the program with the !pgmsav-statement.

Some data you should know about it. The listing is exact 137 pages long - a compiler that especially made to accompany the system reached 87.

The compiler is a real quicky, because it produces 100% machine code and makes the Basic-program run 100 - 150 times faster. It can handle strings and integer array variables and is a must for the programmer, that learned to like programming with the RP-System.

I will now explain, which the statements are, that make your programming life much easier from now on.

I wish you success and good ideas for your coming programs.

Ralph Roeske
Ulft * The Netherlands

Important note:
This program is not protected! The purchaser has the right to make a safety copy for his own use.

Any passing-on, free or as exchange- or selling-good without the consent of Modedeal Co. Ltd. is unlawful and will be prosecuted.

It is allowed to sell a part of the RP-System in the following way and only in that way:

Your program, that is programmed with the RP-System or uses the RP-System, can be sold or passed-on, when it is saved on disk or tape with the PGMSAV-command and you have not changed part of the RP-System.

All programs are tested and bug-free. Programs of that size cannot be bug-free! Modedeal will not be liable for any demage caused by the usage or passing-on of that program.

The Instructions

To see some of the possibilities that the RP-System has to offer, it will be a good idea, to load the first program:

LOAD ":*",8 <Return>
RUN <Return>

The System is initialised. (From now on, I will stop telling you, that you have to press the <Return>-key if you want to send some instructions off. I presume, that you know your C-64 good enough to program it in Basic.)

There are certain demo-programs on the disk and a complete little game, produced in aprox. 2 hrs using the screen-editor to draw the background, converting the hires-bitmap into a characterset, using the sprite-editor to change the RP-System-sprites, draw some new ones and put it all together using the statements that follow in the next chapter.

When you list one of the demos you will see, that all non-commodore-codes begin with a "£"-sign. We did not use reserved basic words because our way to shorten the typing is different from the original one: no letters are typed in shifted. It depends only on the place in the internal codelist whether the system understands one or two or more letters and produces the right statement.

The shortest way to abbreviate a code is told in brackets behind the name of the statement.

£rp (r)

Initalises the System. Bends the interupt-vector, clears the system-variables, switches to the upper 16-K-bank, bits the screen to 50176 and makes the charset ram-based at 51200
It moves also the Basic-Start to $5000, because the RP-System is based at $0801 to nearly $4500 at present.

£rp,h,w
h = screenheight 0=25 lines
1=24 lines
w = screenwidth 0=40 columns
1=38 columns

£rp together with the 2 parameters aditionally alters the screensize.

You should start all your programs with this the £rp-statement.

£xrp (xr)

The opposite of rp. Switches the IRQ off. But the basic-links are still working and the screen- Sprite- and charset-handling is still managed in the upper 16-K-bank.

£transfer (t)

£transfer,f,l,t
f = first memoryplace
l = last memorybyte+1
t = to memoryplace

With transfer you can move or dublicate a free amount of bytes from and to the complete ram of your 64.

Attention!

l must be at least by 1 higher than f, otherwise the system crashes, and you'll have to reload it.
On the other hand you dont have to pay attention whether you move bytes upwards or downwards. £transfer, 20000,22000,22001 is possible and £transfer,20000,22000,19999 too.
If you refer to the places between $a000 and $ffff you always manipulate ram or colormap.

£swap (s)

£swap,f,l,t
f = first memoryplace
l = last memorybyte+1
t = to memoryplace

The same function as transfer; the only difference, after the operation the bytes are not present twice, but swapped.

Swap enables you to have quite a long Basic-program plus variables when you set himem to $8000 (32768) in memory; but also have for instance 25 screens + 25 colormaps + 64 Sprites im memory at a time.

£fill (f)

£fill,f,l,b
f = first memoryplace
l = last memorybyte+1
b = byte

With fill you can erase memory or fill colormaps, screens with a number of equal characters.

£look (l)

£look,p
p = place in memory

After using this statement you can PEEK(680) which byte is in the requested ram-place. Especially made for peeking into the ram under rom.

£partrans (pa)

£partrans,f,l,d,a,t
f = from byte
l = length
d = distance
a = amount
t = to byte

You can pick out parts of memory that are stored in a special way.
Example:
£partrans,57344,64,320,8,51712 fills the charset starting by screencode 64 up to 127 with the upper left 8 by 8 blocks of the bitmap. The first byte of the bitmap is 57344. 64 bytes are equal to 8 character-blocks. 320 bytes are one line of blocks, so the next byte after you picked 64 is 320 bytes from the last line away. Do this 8 times and put the bytes in one line starting at 51200+64*8.

£pretrans (pr)

£pretrans,f,t,l,d,a
f = from byte
t = to byte
l = length
d = distance
a = amount

Is the opposite of the partrans-statement. All the bytes that are in a line are stored in the special way back.

£romram (ro)

$£romram,s
s = sector 0=Basic-rom
1=Operating system

Copies the Rom into the parallel Ram. You may use this, when you dont use Sprites or Bitmap and use for certain purposes a quick-loader or if you use an altered Basic.

£scrnpok (scrnpok)

£scrnpok,l,c,sc
l = line (0-24)
c = column (0-39)
sc= screencodenumber

is the same as the Basicline:
POKE 50176+cv*40+ch where cv = cursor vertical and ch = cursor horizontal.

£design (des)

Only in a Basicprogram
10 £design,sc,v,h
11 /abababab
12 /ccddccdd
. . . . .
18 /aabbccdd

sc= screencode
v = vertical amount
h = horizontal amount
a = "@"
b = "."
c = "*"
d = "^"

£design helps you define characters in normal or multicolormode. Use the redesign-statement and the design-demo to see how to work with this.

Attention!

There must be always the 8 by 8 pixel-rhythm. if v=1 and h=1 then you will have to programm 8 lines with 8 pixels each.
6 and 3 means, you want to define a block of 18 characters. So there must follow 48 lines that begin with a "/" and have 24 signs to represent the bits in each line.
Use "@" and "." for normal and multicolor mode and "*" and "^" only in multicolor.
If you are in multicolor mode, then two equal signs represent one color-pixel.

£redesign (re)

£redesign,f,t
f = from codenumber
t = to codenumber

This statement generates new programlines. It can put half of the charset (128 characters) into programlines to alter the characterset.

£altchar (a)

To view multicolors while defining charsets its possible to change "@*^" into blocks that have the actual multicolor

£charback (ch)

Puts the "@*^" into the charset again.

£blockprt (bloc)

:blockprt,sc,h,v
sc= screencode to start with
h = horizontal amount
v = vertical amount

£cursat,0,0:£blockprt,0,16,16
puts the complete charset in a part of a second onto the screen.

£hires (h)

£hires,m,s
m = mode 0=normal bitmap
1=multicolorbitmap
s = screen 0=no change
1=clear bitmap

Switches the bitmap-mode on.

£text (te)

£text,m,s,c
m = mode 0=normal
1=multicolor
s = screen 0=no change
1=clear screen
c = charset 0=no change
1=caps/blockgraphic
2=caps/small-letters
3=charset from 18432

Switches the textmode on. The charset-parameter 3 requires, that you have loaded a charset f.i. from disk to $4800 or dec.18432.

£scrnsize (scrns)

£scrnsize,v,h
v = vertical 0=25 lines
1=24 lines
h = horizontal 0=40 columns
1=38 columns

Before you use "wrap", "scroll" and "lightwrite" it is optical better to make the screen horizontally smaller.

£old (o)

Makes a hidden program visible. Sets the start-parameters to 20481. Especially useful after undefined statement errors. They work different in the RP-System, because they reset the complete system.

£dir (di)

Brings the directory of the disk onto the screen without erasing the program.

£dir,d
d = devicenumber

Used with a parameter it shows the directory of the disc in the drive with the devicenumber d.

Attention!

There must be a disc in the drive.

£bsav (b)

£bsav n,d,c,f,t
n = name of the file
d = devicenumber
c = channel usually 1
f = from byte
t = to byte+1

Saves the area on tape or disk.

£blod (bl)

£blod n,d,c,t
n = name of the file
d = device
c = channel 1=original adress
0=adress follows
t = to adress. if c=1 then t=0

Loads a program into memory without changing Programpointers in the zeropage.

£merge (me)

£merge n,d
n = name of the file
d = device

Appends a file at the end of a program in memory. The line-numbers must be in order before using merge.

£pgmsav (pg)

£pgmsav n,d,c
n = name of your program
d = device
c = channel (always 1)

Saves your complete Basicprogram including the RP-System (altered!) onto disk or cassette. After you load it in again, run will initialise the RP-System and start your program.

£del (del)

£del,f,t
f = from line
t = to line

Erases the lines that are stated: examples:

£del,20,50 erases line 20 to 50
£del,0,50 erases all lines upto 50
£del,50,0 erases all lines from 50 upwards

£himem (him)

£himem,a
a = page if a<256
    bytenumber if a>255

Sets the Basichimem from the usual 40960 (PEEK(56)=160) to the stated number.

Especially interesting when used with the compiler, because it frees 8192 additional bytes for variables.

£prt.....

£prt
s = secondary adress

Directs all normal prints in a program to the printer device 4. Use 7 for instance to switch on small/capital mode.

£xprt

Closes the channel and directs the printing to the screen again.

£cursat (c)

£cursat,l,c
l = line (0-24)
c = column (0-39)

Sets the cursor. Used before blockprt, bar and print

£dec (dec)

£dec,h
h = hexnumberstring

Only used in directmode and not in Let-statements.

£hex (he)

£hex,d
d = decimal number

Prints the hex-equivalent of a decimal number

Music & Videotext

The easiest way to see how music and videotext are programmed is, to listen and look now into the demo-program music+video.

Generally: You can easy produce every sound you like, without intialising the rp-system. But music, especially polyphonic music, is only possible when directed by the interupt-routines of the RP-System.

Its quite simple to program the 3 voices of the C-64, because you only have to put the notes into Strings as part of your program. Then start it and listen to your own arrangements.

£musicnr (mu)

£musicnr,n,s
n = number of the piece (00-99)
s = string in quotes

n represents the number of one of the 100 possible music-pieces that you must devide a song into, because there can only be a short part of a song in one basic line. This has the advantage, that you can program music just as you program a basic-program: with GOTOs, GOSUBs und RETURNS.

The informations you give are the following:

Notes (C D E F G A B c d e f g a b p # b)
Notelength (1 2 4 8 s z . -)
Control characters (c-s, c-h, c-v, c-p, c-i, &, %, <). Between the informations you can put spaces to make it easier to debug mistakes.

A LIST of the music-demo shows the way how to start. At first the selection of the instruments, level of volume and the height of the notes has to be programmed.

Then the notes, notelength and at the end of a line the "goto"-info, where the music-routine shall continue.

The sysntax:
First a note C to B or an octave higher c to b. A p stands for pause - no sound for a certain time.

Next there follows a "#" or a "b".

If the note should be played half a note higher then the "#"-sign follows the note; f.i. c = cis. To play a note half a note deeper put a "b" direct after the note.

If you want to play the pure note, then the notelength follows directly after the note. You can play full notes (1), half notes (2), quarter notes (4), eighth (8), sixteenth (s) or thirtysecondth (z).

These notelength can be half a length longer by putting a "." behind it (as in the music-books) or can be changed into a triole by putting a "-" after the notelength.
Ctrl-h can be 1 to 9 and is responsible for the base height of the notes. Every step is an octave.

Ctrl-v stands for volume. 1 to 7 are possible inputs.

Ctrl-s changes the speed. 0 to 2 give good results.

The most important statement is the "&"-sign standing for GOTO. Its followed by a two-digit number and tell the system, where to continue. Right inputs are &02 or &92.

The same can be said for the "^"-sign standing for GOSUB. When music-routine reaches that character, it makes a jump to stated musicpart, but as soon as it reaches the "%"-sign (RETURN), it jumps back where it has left and continues.

So you can even program one string with GOSUBs only and jump into all the sublines several times to hear a complete song with the chorus and repetitions only programmed once.

But you can even call subroutines from subrotines as in Basic. 16 GOSUBs per voice (=48) can be open before the first RETURN must be reached - a number you never will need.

To tell the routine, where the music ends put a "<"-sign at the end of the string. The song will end if the repeat-function is not switched on.

You can change the instrument at any point of the music. So you can f.i. program a drum/base-guitar line on one voice only by switching instruments.

Ctrl-i and a two-digit number 01 to 16 points at the next instrument that shall be switched on.

£mplay (mp)

£mplay,n,v(,n,v)(,n,v)
n = musicnumber (0-99)
v = voice (1-3)

You start the music with this statement. It is obvious, that the program must have passed the lines with the musicstrings, envelope and filtersettings before it reaches the mplay-code.

With the mplay-statement you tell the routine where to start the song (musicno. for each voice) and which voice shall play which musicpart. You start all 3 or 1 or 2 voices at a time. They start exact at the same moment and if you have programmed the musicstrings right, they will always play exact parallel.

Although you will usally start the 3 voices at 3 different musicstrings, it is possible, that all 3 voices play the same string probably with 3 different instruments (GOSUB after instrument-setting)

£xmplay (xm)

£xmplay,v
v = voice 0=all 3 voices
1-3=one voice

Switches one or all voices off.

£mrepeat (mr)

Located somewhere behind the mplay-command, this brings the option in operation, that, when the music-routine reaches the end-sign "<", it will play the song over and over again from the beginning.

£xmrepeat (xmr)

£xmrepeat,v
v = voice (as xmplay)

Next time, the music-routine reaches the end-sign "<" it will stop playing the voice or all 3 voices.

£mfade (mf)

£mfade,f
f = fading time (1-255)
Its a pity that the C-64 has only one volumecontrol for all three voices. So all three voices fade equal after this command.

£playsound (p)

£playsound,v,n,i
v = voice 1-3
n = note 1-120
0=frequency (see envelope)
i = instrument 1-16

To "construct" an instrument you can only try and try and try. The playsound-statement helps you to change envelope and filtersettings, because it lets you hear the sounds you create as often as you need it.

£envelope (e)

£envelope,i,a,d,s,r,w,p,f,q
i = instrumentnumber 1-16
a = attack 0-15
d = decay 0-15
s = sustain 0-15
r = release 0-15
w = waveform 0=triangle
1=sawtooth
2=rectangle
3=white noise
4=ringmodulation
p = pulswidth 1-4095
f = filternumber 1-16
q = frequency 1-65535

All the sounds and instruments you need are set with this command: barking, galloping, trumpet, violin, etc.

16 sounds can be in memory at a time. Since you can only try, here are some instruments that Commodore recommend in its handbook for the C-128.

Piano	n,0,9,0,0,2,1536
Accordion	n,12,0,12,0,1,0
Circus organ	n,0,0,15,0,0,0
Drums	n,0,5,5,0,3,0
Flute	n,9,4,4,0,0,0
Guitar	n,0,9,2,1,1,0
Cembalo	n,0,9,0,0,2,512
Organ	n,0,9,9,0,2,2048
Trumpet	n,8,9,4,1,2,512
Xylophone	n,0,9,0,0,0,0

As you see the pulswidth is only recommended, when rectangle sounds (2) is used.

If you put a "0" for filter then no filters will be activated.

q for frequency is very important, when you call that special sound from the "spritesound" or if you test sounds with the playsound-statement. Clever settings let you produce motorsounds, helicopters etc. very easily

£filter (filt)

£filter,f,c,l,b,h,r
f = filternumber 1-16
c = cutoff frequency 0 -2047
l = lowpass 0-1
b = bandpass 0-1
h = highpass 0-1
r = resonance 0-15

This command sets the registers 21-24 in the SID. The best is, to program a little basic-program which contains the envelope, filter and playsound-statement. Functionkeys and several others may alter the settings and you can hear every change you made.

Complete view

Instrumentsetting:	i00-16
Octaveheight	: h0-9
Volume	: v0-7
Speed	: s0-2
GOTO	: &00-99
GOSUB	: ^00-99
RETURN	: %
End	: <
Notes	: CDEFGABcdefgab
Half tone higher	: #
Half tone deeper	: b
Tonelength	: 1 2 4 8 s z
1 1/2 length	: .
Triole	: -

Note: underlined letters are controlcodes. Press the controlkey and the letterkey at the same time.

£vidtxt (v)

Switches to highresolution-bitmap-mode and prepares the videotext-variables for the comming text.

You can have lightwriting and videotext at the same time.

£vidprt (vidp)

£vidprt,t
t = text between quotes
or in stringvariables

Works in most cases as the Basic-statement PRINT. But all text appears on the visible bitmap and there are some new control-commands.

clr (shift home) clears the screen as in Basic-print.

home puts the cursor into the home-position.

Cursor as in Basic-print CRSR up, down, left and right

rvs on as in Basic-print (reverse on)

rvs off as in Basic-print

ctrl-b Setting of the backgroundcolor. Each following character has the same backgroundcolor. You press ctrl-b and then the color as in the Basic-printcommand.

ctrl-p pencolor. All following characters are printed in the stated color.

ctrl-d double height. Displays the characters in vertical doubled size. You must position the cursor at the right line before you print or you may bomb the system.

ctrl-n normal height

ctrl-a alternative characterset. The following 5 digits (5!!!) tell the system, where the alternative charset is located in memory. The original set is at 51200. Please dont use the Ram under the Roms this time.

ctrl-c position cursor. The following 4 digits hold the new cursor position. c1203 means, that printing continues at row 12 column 3.

ctrl-x erase line in which the cursor is positioned.

ctrl-h hold position. Normally when the string ends, the system simulates a carriage return. If the string ends with ctrl-h no carriage return is executed.

Lightwriting, Wrap and Scroll

The RP-System is not only used to program games. Some professionel programs, used f.i. in shop-window advertising, can be created.

£lightwrite (li)

```
£lightwrite,f,m,t,c
f = first line (0-17)
m = mode 0=blockgraphics
         1=light-dots
t = textpointer
c = color
```

Text, that should appear as a light-band, must be present in screen-code. That means, the easiest is, to print it invisible (printcolor is backgroundcolor) on the screen and then tranfer it to the textarea to save the complete text. If you print revers, the lightwriting will be reverse, too.

£lightcol (lightc)

```
£lightcol,c
c = color 1-15
```

You can change the color at any beginning of a new letter.

The Speed is changed with the command £wrapspeed and it is switched off with £xwrap.

You can have bitmap-mode, sprites, music and lightwriting at the same time.

£wrap (w)

```
£wrap,f,l
f = first line (0-24)
l = last line (0-24)
```

Part of the screen rolls round, that means the characters that leave the screen at the left side, appear on the right side again.

The difference between f and l should not be bigger than 9 or 10 and l must be equal or greater than f.

Especially when you have reduced the screensize vertically, you need to adjust the rasterline. You do that with £wraprast.

£wrapspeed (wraps)

```
£wrapspeed,s
s = speed (0-8)
```

Adjusts the speed at lightwrite, wrap and scroll.

£xwrap (xw)

Switched Wrap, lightwrite or scroll off.

£wraprast (wrapr)

```
£wraprast,f,s
f = first rasterline (40-65)
s = second rasterline (48-73)
```

The general setting is 49 and 57 but you can put it wherever you need it.

£scroll (scro)

```
£scroll,f,l,c,cp
f = first line (0-24)
l = last line (0-24)
c = characterpointer
cp= colorpointer
```

As in Wrap you set the first and last line of the screen that shall scroll. Aditionally the system needs the information, where it has to pick the next characters and the matching colours.

The characters have to be in memory in a way that no calculating is needed to put them onto the screen. So if the scrolling band is 8 lines wide, than there must be the 8 characters that will appear at the right side of the screen in a row in memory.

The end of the chain must be 0, what means, that you cannot use the character with the screen-code 0.

Clock and Bargraphic

In many games the exact time is recommended. The C-64 has exact timers. So one should use them.

Results of games are usually a line of numbers. Try something new. Put the results onto the screen in better optical view.

We included time and bargraphic in the system.

£setclock (se)

```
£setclock,"000000",sl,sc,c
"000000" time in 6 digits as TI$
sl= screenline 0-24
sc=screencolumn 0-30
c = color 1-16
```

Starts the internal clock.

£lap (la)

The clock stops on the screen but continues intern. The next lap puts the right time onto the screen.

£clockpara (cl)

```
£clockpara,sl,sc,c
sl=screenline (0-24)
sc=screencolumn (0-30)
c = color (1-16)
```

m2= multicolor2 1-16
0=no change

This command replaces the pokes into the VIC and sets the multicolors of the multicolorsprites you use.

£scrncol (sc)

£scrncol,bc,b,m1,m2
bc= backgroundcolor 1-16
0=no change
b = bordercolor 1-16
0=no change
m1= multicolor1 1-16
0=no change
m2= multicolor2 1-16
0=no change

This command makes the same as the previous. Instead of the sprite, the screenmulticolors, background and border will be set.

£sprsound (sprso)

£sprsound,n,i,v,tp
n = spritenumber 0-7
i = instrumentnumber-1 (0-15)
v = voice 1-3
tp= timepattern 1-255

This command is the one, that is responsible for sound, "played" by the sprites!

Spritenumber, instrument and voice are sent to the RP-System, as explained at the music- and sprite-commands.

Timepattern is unique for this command. It means, that you can tell the System, when a sound should be audible. When a spritesequence consists of 8 patterns, then each bit in the timepattern represents one pattern of the sequence.

Lets explain that with an example: supposing, you have drawn an 8-pattern-hopper, that touches the ground on the first pattern, then the timepattern = 1. If it touches the ground on the 1st and on the 5th pattern, then the timepattern = 17. You can calculate the timepattern with the following formula:

tp=2^pattern(+2^pattern)(+2^pa...)

£xsprsound (sprs)

£xsprsound,n
n = spritenumber

Switches the spritesound off.

£synchro (sy)

£synchro,s1,s2
s1= sprite 1
s2= sprite 2

As the name says, this statement makes two sprites move synchronous, even if they where initialized independent of each other. Imagine you let in a game an indian jump on his running horse. That would be a wild scene when the horse jumps up while the indian (programmed to move synchronous) is in the down-phase. £synchro puts this right.

You can let all 8 sprites run synchronous if put then all in a chain or if you use as s1 (sprite1) allways the same number.

£xsynchro (xsy)

£xsysnchro,s
s = synchrosprite

When it is not more needed, that sprites move synchronous; or once they are synchronized, you can switch the synchro-routine off. Put in the number of the sprite, that was used as Sprite 2 in the previous command.

£sprtrans (sprt)

£sprtrans,f,a,p
f = from byte
a = amount 1-128
p = pattern 128-255
You cannot load spritedata from cassette or disc directly into the ram unter the operating system rom. So you have to load spritedata to a free place and then transfer it to the place, where the RP-System needs them. This is done with £sprtrans.

128 sprites can be used and placed from $e000. This is the maximum amount.

Most time you will use the free ram between 40960 ($a000) and 49152 ($c000) to load hires-screens and spritedata in.

£joyfreq (j)

£joyfreq,s1,s2
s1= stroke 1 1-8
s2= stroke 2 1-8

In many modern games (Olympics f.i.) you have to move the joystick as fast as possible, or in a certain rhythm. In other words, you need the joystickmovingfrequence if you want to program a similar game. You switch this feature in the RP-Sytem on and can from then on PEEK(689) or PEEK(690) the frequence, the joystick is moved. For s1 and s2 you will usually put in opposite numbers (1 and 5, 3 and 7 or 2 and 6). See chapter joystick and spritedata at the end of the instructions.

£xjoyfreq (xj)

This command switches the feature off.

£firejmp (fir)

£firejmp:gotoj1:gotoj2
j1= jumpaddress port 1
j2= jumpaddress port 2

The program branches as soon as a firebutton of one of the 2 joysticks is pressed to the linenumber given with the goto(!!!)-statements.

This is especially implemented for usage with the RPS-Compiler, but if you program it right, then its a good thing to use it from basic as well.

£hgrchar (hg)

£hgrchar,cn
cn= characternumber 0-255

This commands changes a hiresolution bitmap into a characterset and the textscreen belonging to it. The colormap will remain the same and can be used for both hires- and text-screen.

If you want to keep the first 64 letters and characters and the bitmap consists of a big amount of blocks that are equal, then you put in 64 for cn.

The program need the bitmap starting from $e000 (57344). The characterset, that will be created is located at $a800 (43008) and ends at $b000 (45056). The textscreen that will hold the new screencodes starts at $a000 (40960) and ends at $a3e8 (41960).

If the system crashes and the routines are not destroyed and the LIST-command produces the RP-System-title the a RUN and £old can bring your program back.

Attention!

A jump into a non-existend linenumber does not produce an undef'd statement error anymore. Instead the system is set back and should recover when you try the list/run/£old as described in the last paragraph.

Joystick and Spritedata

Everytime (60 times a second), when the IRQ-routine is called, the actual data for sprites and joystick are put into reserved registers. So you can get them with PEEK (or DEEK from the compiler).

The PEEKs are:
Direction joystick 1: PEEK(681)
Direction joystick 2: PEEK(682)
The values you get are 0-8 and you interpret them as follows:

```
      1
   8  2
 7  0  3
   6  4
      5
```

The following registers hold a 1, 0 or 255 depending in which direction the joystick is pressed.

```
Joystick1 X-sgn: PEEK(685)
          y-sgn: PEEK(686)
Joystick2 X-sgn: PEEK(687)
          y-sgn: PEEK(688)
```

Is the firebutton pressed, then the registers 683 and 684 change their values.

Button1: PEEK(683)
Button2: PEEK(684)

The spritedata are put into the registers 735 - 766

```
          xlo/xhi   ylo/yhi
Sprite 0: 735/736   737/738
Sprite 1: 739/740   741/742
Sprite 2: 743/744   745/746
Sprite 3: 747/748   749/750
Sprite 4: 751/752   753/754
Sprite 5: 755/756   757/758
Sprite 6: 759/760   761/762
Sprite 0: 763/764   765/766
```

The C-64-memory is used as follows:

```
01204-02048 System-variables
02049-18432 RP-System
18432-20480 free (base-char-set)
20480-40960 Basic-program
40960-49152 free
49152-50176 spritedata
50176-51200 screen+sprites
51200-53248 characterset
53248-57344 VIC SID I/O
57344-65535 128 Sprites
            or Hires-bitmap
```

There are some demos and a free game on the disc. The little game demo was produced in 20 minutes.

The Grand National needed 2 hrs, using the SPRITE SEQUENZ MAKER and the SCREEN-EDITOR from HIGH SPEED SOFTWARE.

Please look into the demo-programs to get easier familiar with the system.

If you have advice or have noticed bugs, then please write to:

Modedeal Co. Ltd.
1, Coombe House
Hartland Road
Addlestone Surrey
KT15 1JU
England